The Complexity of Order Dependency Inference

Jaroslaw Szlichta, Parke Godfrey, Jarek Gryz and Calisto Zuzarte

Technical Report CSE-2012-05

August 27 2012

Department of Computer Science and Engineering
4700 Keele Street, Toronto, Ontario M3J 1P3 Canada

The Complexity of Order Dependency Inference

Jaroslaw Szlichta
York University in Toronto &
IBM Toronto
Center for Advanced Studies
jszlicht@cse.yorku.ca

Parke Godfrey
York University in Toronto &
IBM Toronto
Center for Advanced Studies
godfrey@cse.yorku.ca

Jarek Gryz
York University in Toronto &
IBM Toronto
Center for Advanced Studies
jarek@cse.yorku.ca

Calisto Zuzarte
IBM Toronto Laboratory
calisto@ca.ibm.com

ABSTRACT

Dependencies play an important role in database theory. We study *order dependencies* (ODs)—and *unidirectional* order dependencies (UODs), sub-class of ODs—which describe the relationships among lexicographical orderings of sets of tuples. We investigate the *inference problem* for order dependencies. We establish the following: (i) a sound and complete chase procedure for ODs for testing logical implication demonstrating its decidability, but prove the undecidability of the inference problem for ODs with inclusion dependencies; (ii) a proof of co-NP-completeness for the inference problem for the subclass of UODs; (iii) a proof of co-NP-completeness for the inference problem of functional dependencies (FDs) from ODs in general, but demonstrate linear time complexity for the inference of FDs from UODs; and (iv) a sound and complete inference algorithm for sets of UODs over a *natural* domain.

1. INTRODUCTION

Understanding the semantics of data is important, both for data quality analysis and knowledge discovery. While the relational data model is *set* based and does not concede the concept of *order*, ordered streams nonetheless play important roles in relational systems. SQL allows one to specify by its order-by clause the answer "set" to be returned in the specified order. Ordered streams are prevalent in query plans between operators to provide efficient evaluation. A query optimizer must reason extensively over *interesting orders*, while building efficient query plans [17].

Order for a tuple stream can be semantically specified via the attributes as by SQL's order-by clause. The *order specification* order by year desc, name asc requires that the tuple stream be sorted by year in descending order and, within each year group, by name in ascending order. This is a *lexicographical* ordering, a nested sort. (Note there could be many *total orders* that satisfy this specification: the tuples within any year-name partition may be ordered freely.)

Order dependency is the semantic relationship amongst order specifications. An *order dependency* states a relationship between two order specifications. Say that we knew the OD that id asc *orders* year asc, name asc. Then we would be assured that any tuple stream ordered by id asc would also necessarily be ordered by year asc, name asc. (Note the converse is *not* necessarily assured: if the stream were ordered by year asc, name asc, it still might not be ordered by id asc. This is because the tuples in a given partition of year-name might fail to be ordered by id asc.)

The concept of order dependency is closely related to that of *functional dependency*. Indeed, we shall show that order dependency *subsumes* functional dependency. If id asc *orders* year asc, name asc, then the functional dependency (FD) that id *functionally determines* year and name must hold. ODs convey additional semantic information, of course: that of order. Furthermore, working with ODs is more complex than working with FDs, because the *sequence* of the attributes in order specifications matters. ODs are specified with respect to *lists* of attributes, whereas FDs are specified with respect to *sets* of attributes.

Order dependency has been studied before with respect to lexicographical orders [15, 19], and with respect to other order definitions [9, 10]. We restrict our focus to order dependencies over lexicographical orders. While lexicographical order depencency has been studied before, it has not been well understood. The *inference problem* is to answer whether an OD is logically entailed by a set of ODs. The *decidability* and the *complexity* of the inference problem for (lexicographical) order depencency have heretofore not been known. We address this in this work.

```
select D.year, D.trimester, D.quarter,
       D.month, D.day
       sum(S.sales) as total
       count(*) as quantity
from date_dim D, sales S
where S.date_id = D.date_id and
      D.year between
         2001 and 2004
      and sum(S.sales) > 10000
group by D.year, D.trimester, D.quarter,
         D.month, D.day
order by D.year, D.trimester, D.quarter,
         D.month, D.day;
```

Query 1: Eliminating trimester and quarter.

Consider the SQL query in Query 1 over a data ware-

.

house schema as in [12]. The fact table sales has a foreign key S.date_id which references the dimension table date_dim. *Date* is captured in a hierarchical manner by attributes year, quarter or trimester, month, and day. The values of the attribute quarter divides year into four three-month periods, while those of trimester divides it into three four-month periods.

Let there be a B+ tree index for date_dim on year, month, day. The query optimizer may not employ this index to evaluate either the group-by or the order-by for the query in Query 1, because their specifications do not *match* the index's search key.

Of course, it is clear that month functionally determines quarter and trimester. So partitioning by year, trimester, quarter, month, day is the same as just by year, month, day. In fact, optimizers today would eliminate trimester and quarter from the group-by via the relevant FDs [17], and then employ the index for the group-by operation.[1]

The FD that month → quarter, trimester is not logically sufficient to optimize the order-by operation, however. One would need the additional semantic information of an OD that year, month, day *orders* year, trimester, quarter, month, day. This and similar subtleties cause the optimizer to miss opportunities to use indexes and to pipeline operations. Expensive operations as sort are added to a query plan, even when the data is already sorted properly. By incorporating reasoning over ODs into the optimizer—as has already been done for reasoning over FDs—many new optimizations would be possible [20, 21]. The ordered stream by year, month, day then could satisfy *both* the group-by and the order-by operations *on-the-fly*.

The contributions of this work are as follows.

1. *Fundamentals.* (Section 2.)
   (a) Introduce a theoretical framework for ODs.
   (b) *Unidirectional ODs and Axiomatization.*
      i. Introduce a prevalent sub-class of ODs, *unidirectional order dependencies* (UODs), and prove the sub-class is proper.
      ii. Present a sound and complete axiomatization for UODs [19], sound for ODs.
   (c) Demonstrate how ODs can be used effectively in optimization, by putting ODs into a canonical form to enable reasoning over ODs in the query optimizer.
2. *Decidability.* (Section 3.)
   (a) Demonstrate the decidability of the OD inference problem via a *sound* and *complete chase procedure* for ODs. (Preliminary work appears in [18].)
   (b) Prove that the inference problem for ODs with inclusion dependencies is undecidable.
3. *Complexity.* (Section 4.)
   (a) Establish that the inference problem for UODs is co-NP-complete.
   (b) Establish that the inference problem of inferring FDs from ODs is also co-NP-complete, but that it is only linear for the case of FDs over UODs.
4. *Inference Procedures.* (Section 5.)
   (a) Identify a restricted, *natural* domain, the *order-compatible transitive domain*, which makes reasoning over ODs simpler.[2]
   (b) Devise an efficient, polynomial inference procedure for testing implication of ODs over the transitive domain that is sound and complete.

In Section 6, we discuss related work. In Section 7, we conclude and consider future work.

Table 1: Notational conventions.

- **Relations**
  - $\mathbf{R}$ denotes a *relation*.
  - $\mathbf{r}$ denotes a specific *relation instance* (*table*).
  - A, B and C denote *attributes*.
  - s and t denote *tuples*.
  - t_{A} denotes the value of attribute A in tuple t.
- **Sets**
  - $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ denote *sets*.
  - $t_{\mathcal{X}}$ denotes the *projection* of tuple t on $\mathcal{X}$.
  - $\mathcal{XY}$ is shorthand for $\mathcal{X} \cup \mathcal{Y}$.
- **Lists**
  - $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ denote *lists*. (Note $\mathbf{X}$ could represent the empty list, [].)
  - $[\mathsf{A}, \mathsf{B}, \mathsf{C}]$ denotes an explicit list. $[\mathsf{A} \mid \mathbf{T}]$ denotes a list with *head* A and *tail* $\mathbf{T}$.
  - $\mathbf{XY}$ is shorthand for $\mathbf{X} \circ \mathbf{Y}$ ($\mathbf{X}$ concatenate $\mathbf{Y}$).
  - set $\mathcal{X}$ denotes the *set* of elements in *list* $\mathbf{X}$. Anyplace a set is expected but a list appears, the list is *cast* to a set; e.g., $t_{\mathbf{X}}$ denotes $t_{\mathcal{X}}$.

2. FUNDAMENTALS

We first set the notational conventions and definitions of ODs and UODs. Then we introduce an axiomatization which is sound and complete for UODs and sound for ODs. In the end, we present where ODs arise and how they can be used for optimization.

2.1 Framework

We adopt the notational conventions as in Table 1. We model *order specification* as provided by SQL's order-by clause for specifying lexicographical orderings.

Definition 1. (order specification) An *order specification* is a list of *directionality-marked attributes* (or *marked* attributes, for short).

There are two directionality operators: asc and desc, indicating *ascending* and *descending*, respectively. Each operator is unary, applies over an attribute, and is written postfix; e.g., A asc and B desc. As shorthand notation, we write $\overrightarrow{\mathsf{A}}$ and $\overleftarrow{\mathsf{A}}$ for A asc and A desc, respectively. As further notational shorthand, we merge the top arrows for adjacent attributes that are marked with the same directionality; e.g., $[\overrightarrow{\mathsf{AB}}\overleftarrow{\mathsf{C}}]$ denotes $[\overrightarrow{\mathsf{A}}\overrightarrow{\mathsf{B}}\overleftarrow{\mathsf{C}}]$.

In any context an *order specification* is expected but a list of (unmarked) attributes appears, the list is cast to the order specification with each attribute marked as asc; e.g., $[\mathsf{A}, \mathsf{B}, \mathsf{C}]$ is cast to $[\overrightarrow{\mathsf{A}}, \overrightarrow{\mathsf{B}}, \overrightarrow{\mathsf{C}}]$.[3]

The order specification $\mathbf{X}$ defines an algebraic relation '$\preceq_{\mathbf{X}}$'. The operator '$\preceq_{\mathbf{X}}$' defines a *weak total order* over any set of tuples.

[1] IBM DB2 incorporates such rewrites.

[2] A domain is *restricted* if an additional order property is guaranteeed over the schema. The *order-compatible transitive domain* is quite natural in that it is intuitive, it holds for all real-world business domains that we have encountered, and it can easily be verified whether it holds.

[3] *Ascending* is the default for SQL in order-by for any attributes for which directionality is not explicitly indicated.

Table 2: Relational instance $\mathbf{r}$.

#	A	B	C	D	E
s	1	4	4	6	3
t	2	3	4	6	4

Definition 2. (algebraic relation '$\preceq_{\mathbf{X}}$') Let $\mathbf{X}$ be a list of marked attributes. For two tuples r and s (over a schema containing the attributes in $\mathbf{X}$), $r \preceq_{\mathbf{X}} s$ *iff*

- $\mathbf{X} = [\overrightarrow{\mathsf{A}} \mid \mathbf{T}]$ and $r_{\mathsf{A}} < s_{\mathsf{A}}$; or
- $\mathbf{X} = [\overleftarrow{\mathsf{A}} \mid \mathbf{T}]$ and $r_{\mathsf{A}} > s_{\mathsf{A}}$; or
- $\mathbf{X} = [\overrightarrow{\mathsf{A}} \mid \mathbf{T}]$ or $\mathbf{X} = [\overrightarrow{\mathsf{A}} \mid \mathbf{T}]$, $r_{\mathsf{A}} = s_{\mathsf{A}}$, and $r \preceq_{\mathbf{T}} s$; *or*
- $\mathbf{X} = [\,]$.

Let $r \prec_{\mathbf{X}} s$ *iff* $r \preceq_{\mathbf{X}} s$ but $s \not\preceq_{\mathbf{X}} r$.

We now define *order dependencies*.

Definition 3. (order dependency) Let $\mathbf{X}$ and $\mathbf{Y}$ be lists of marked attributes. $\mathbf{X} \mapsto \mathbf{Y}$ denotes an *order dependency* (OD), read as $\mathbf{X}$ *orders* $\mathbf{Y}$. $\mathbf{XY} \leftrightarrow \mathbf{YX}$, read as $\mathbf{X}$ and $\mathbf{Y}$ are *order equivalent*, *iff* $\mathbf{X}$ *orders* $\mathbf{Y}$ and $\mathbf{Y}$ *orders* $\mathbf{X}$. Let $\mathbf{R}$ be a relation (over a schema that contains the attributes that appear in $\mathbf{X}$ and $\mathbf{Y}$), and let $\mathbf{r}$ be a relation instance of $\mathbf{R}$. Table $\mathbf{r}$ *satisfies* $\mathbf{X} \mapsto \mathbf{Y}$ ($\mathbf{r} \models \mathbf{X} \mapsto \mathbf{Y}$) *iff*, for all $s, t \in \mathbf{r}$, $r \preceq_{\mathbf{X}} s$ implies $r \preceq_{\mathbf{Y}} s$. The OD $\mathbf{X} \mapsto \mathbf{Y}$ is said to *hold* for $\mathbf{R}$ ($\mathbf{R} \models \mathbf{X} \mapsto \mathbf{Y}$) *iff*, for each admissible relational instance $\mathbf{r}$ of $\mathbf{R}$, table $\mathbf{r}$ satisfies $\mathbf{X} \mapsto \mathbf{Y}$.

Example 1. *Let* $\mathbf{r}$ *be a relation instance over* $\mathbf{R}$ *with attributes* A, B, C, D, *and* E, *as shown in Table 2. Note* $\mathbf{r} \models \overrightarrow{\mathsf{ACD}} \mapsto \overrightarrow{\mathsf{EB}}$, *but* $\mathbf{r} \not\models \overrightarrow{\mathsf{ACD}} \mapsto \overrightarrow{\mathsf{BE}}$. *Also note* $\mathbf{r} \models \overleftarrow{\mathsf{CA}} \mapsto \overrightarrow{\mathsf{B}}\overleftarrow{\mathsf{DE}}$, *but* $\mathbf{r} \not\models \overleftarrow{\mathsf{CA}} \mapsto \overrightarrow{\mathsf{EB}}\overleftarrow{\mathsf{D}}$.

Order dependencies can be prescriptive statements on the relation, as can be functional dependencies. That is, they can be used as a type of integrity constraint to prescribe which instances are admissible.

We introduce one additional order relation, *order compatibility*, to capture the notion that if two lists $\mathbf{X}$ and $\mathbf{Y}$ are order compatible, there is a way to order the tuples of any table so the ordering satisfies both the order spefications $\mathbf{X}$ and $\mathbf{Y}$. $\mathbf{X}$ and $\mathbf{Y}$ may be order compatible without either $\mathbf{X} \mapsto \mathbf{Y}$ or $\mathbf{Y} \mapsto \mathbf{X}$. At first glance, this might seem surprising. A degenerate case demonstrates this quickly, however. The empty order spefication, [ ], is order compatible with *any* order spefication. (Any ordering of tuples satisfies the order spefication [ ].)

Interestingly, order compatibility does not add expressiveness over order dependencies as already introduced. Indeed, we can define it directly as an OD of specific form. Because the concept proves invaluable, however, for reasoning about ODs, we introduce it explicitly for this purpose.

Definition 4. (order compatible) Two order spefications $\mathbf{X}$ and $\mathbf{Y}$ are *order compatible*, denoted as $\mathbf{X} \sim \mathbf{Y}$, *iff* $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

2.2 Unidirectional ODs and Axiomatization

We accommodate bidirectionality (asc and desc) in order specifications for generality's sake, and because SQL's order-by clause does. Of course, marking attributes adds complication. It is reasonable to ask whether this bidirectionality adds expressiveness, and if so, whether the added expressiveness is useful.

One can consider a simplified version of ODs for which we remove this bidirectionality. Call a set of ODs *unidirectional* in which any given attribute appears in the ODs either marked as asc or as desc, but not both. Without loss of generality, one can consider just sets of ODs in which all attribute occurrences are marked as asc. Call an individual OD in which all attributes are marked as asc a *unidirectional* order dependency (UOD). This restriction to just ascending has the advantage that one can verify that a set of ODs is unidirectional by verifying in isolation that each OD is unidirectional.

1. *Reflexivity.*
$$\mathbf{XY} \mapsto \mathbf{X}$$
2. *Prefix.*
$$\frac{\mathbf{X} \mapsto \mathbf{Y}}{\mathbf{ZX} \mapsto \mathbf{ZY}}$$
3. *Normalization.*
$$\mathbf{WXYXV} \leftrightarrow \mathbf{WXYV}.$$
4. *Transitivity.*
$$\frac{\mathbf{X} \mapsto \mathbf{Y} \quad \mathbf{Y} \mapsto \mathbf{Z}}{\mathbf{X} \mapsto \mathbf{Z}}$$
5. *Suffix.*
$$\frac{\mathbf{X} \mapsto \mathbf{Y}}{\mathbf{X} \leftrightarrow \mathbf{YX}}$$
6. *Chain.*
$$\frac{\mathbf{X} \sim \mathbf{Y}_1 \quad \forall_{i \in [1,n-1]} \mathbf{Y}_i \sim \mathbf{Y}_{i+1} \quad \mathbf{Y}_n \sim \mathbf{Z} \quad \forall_{i \in [1,n]} \mathbf{Y}_i\mathbf{X} \sim \mathbf{Y}_i\mathbf{Z}}{\mathbf{X} \sim \mathbf{Z}}$$

Figure 1: Axioms for UODs [19].

Definition 5. (unidirectional order dependency) An order dependency is *unidirectional* when all attributes within it are marked asc. In contrast, call an order dependency which has attributes marked both as asc and as desc a *bidirectional* order dependency (BOD).

In [19], we studied UODs and provided a *sound* and *complete* axiomatization for them. The inference rules of the axiomatization are shown in Figure 1. (They provide insight in later sections.)

Theorem 1. [19] *(soundness and completeness) The set of the axioms from Figure 1 is sound and complete over UODs.*

Syntactically, UODs are a sub-class of ODs, by definition. It is trivial to show that the axiomatization in Figure 1 is *sound* for ODs.

Corollary 2. *(soundness over ODs) The set of the axioms from Figure 1 is sound over ODs.*

Does the addition of BODs, however, add expressive power? If one could provide a universal *translation* for any set of ODs (UODs and BODs) to a *semantically equivalent* set of UODs, then one could say that bidirectionality does not add expressiveness.[4]

With BODs, ODs are more expressive than UODs. No such translation exists. We prove this by demonstrating that the sound and complete axiomatization in Figure 1 is *not* complete for ODs.[5]

Theorem 3. *(incomplete for ODs) The set of the axioms from Figure 1 is not complete over ODs.*

Proof

Consider the set $\mathcal{M}$ of $[\overrightarrow{\mathsf{A}}] \mapsto [\overrightarrow{\mathsf{B}}]$ and $[\overrightarrow{\mathsf{A}}] \mapsto [\overleftarrow{\mathsf{B}}]$. From first principles, it is simple to show that $\mathcal{M} \models [\,] \mapsto [\overrightarrow{\mathsf{B}}]$. None of the axioms *reduce* the left-hand side of an OD (besides *Normalization*, which does not apply here). $\mathcal{M} \models [\,] \mapsto [\overrightarrow{\mathsf{B}}]$

[4]To be *semantically equivalent*, there would need to be a bijection of ODs that can be inferred from the set of ODs and UODs that can be inferred from the corresponding set of UODs, modulo the translation.

[5]To find a sound and complete axiomatization for ODs is an open question.

cannot be proved from the axioms. □

Therefore, the class of ODs are more expressive than its sub-class, the class UODs. Semantically, UOD is a proper sub-class of OD. The next question is the cost for this extra expressiveness. Is the inference problem for ODs decidable? This is resolved in Section 3. What is the complexity of the inference problem for UODs? Assuming decidability, is the complexity of the inference problem for ODs higher? These questions are resolved in Section 4.

2.3 Optimization with ODs

We describe how order, and order dependencies, can be used for query optimization. Order dependencies can be declared as an *integrity constraints*. A database administrator who knows well the semantics of the database could specify ODs as constraints for it. However, OD optimization techniques are also applicable even when the database has no declared ODs. Order dependencies can be implied by queries' semantics. For example, if there is a predicate $\mathsf{A} = \mathsf{B}$, then an OD $\mathsf{A} \leftrightarrow \mathsf{B}$ is satisfied within the scope of the query. ODs also arise through SQL functions and algebraic expressions. For instance, UODs $[\mathsf{d\_date}] \mapsto [\mathsf{year(d\_date)}]$ and $[\mathsf{d\_date}] \mapsto [\mathsf{d\_date} + \mathsf{30\ days})]$ hold [21].

Order is not relevant on the *logical* side in the relational model. (There are data models such as XML, for which order is part of the model.) Order is important on the *physical* side. Order plays a significant role in storage, indexes, pipelining, and keeping interesting orders [17].

We motivate *order dependencies* in analogy to *functional dependencies*. ODs are to order-by as FDs are to group-by. ODs might be used in query optimization [20, 21] just as FDs have been before [17]. In [20], we showed how ODs can provide significant performance improvement by eliminating join from query plans in a data warehouse environment. We built a prototype in IBM DB2 V.9.7 and performed experiments over the TPC-DS benchmark to demonstrate the significant efficiency of the approach. In [21], we showed how ODs between columns and SQL functions or algebraic expressions over columns can bring benefits for queries that involve join, order-by, group-by, and distinct statements.

Assume that we are aware of an OD $\mathbf{X} \mapsto \mathbf{Y}$. Therefore, a query with `order by` $\mathbf{Y}$ can be rewritten with `order by` $\mathbf{X}$. Note that the original and rewritten query are not semantically equivalent, unless $\mathbf{X} \leftrightarrow \mathbf{Y}$. The rewritten query satisfies the order of the original query, but not necessary vice versa. That is, order *equivalency* is not required for correct query rewrites. Directional order dependencies ($\mathbf{X} \mapsto \mathbf{Y}$) are sufficient to provide wide variety of query rewrites.

The important role of *order* was examined in [17]. When it is known that the orders are *order equivalent* as defined in this work, one interesting order can be replaced by another. However, this technique relies on FD information and does not incorporate ODs. The authors of [17] designed the algorithm *Reduce Order*, which scans the interesting order list backwards to test if any of the attributes can be eliminated using FD information. We extend this algorithm by iterating through the list additionally testing whether the list without the attribute being currently considered *orders* the full list. (Call this *Reduce Order OD*.) If the OD holds, then the attribute can be removed from the current list. When sorting is required, the reduced version of an interesting order $\mathbf{I}$ provides a smaller number of sorting columns, which reduces cost.

Algorithm 1 Reduce Order OD

Input:
order specification $\mathbf{I} = [\mathsf{I}_0, \mathsf{I}_1, ..., \mathsf{I}_{n-1}]$ and
a set of ODs $\mathcal{M}$.

Output:
The normalized interesting order $\mathbf{I}$.

1: Rewrite $\mathbf{I}$ in terms of each column's equivalence class head.
2: Comment: scan $\mathbf{I}$ backwards
3: **for** $i \leftarrow n-1$ **to** 0 **do**
4: Let $\mathsf{B} = \{\mathsf{I}_0, ..., \mathsf{I}_{i-1}\}$
5: **if** $\mathsf{B} \rightarrow \mathsf{I}_i$ **then**
6: Remove I_i from $\mathbf{I}$
7: **else if** $[\mathsf{I}_0, ..., \mathsf{I}_{i-1}, \mathsf{I}_{i+1}, ..., \mathsf{I}_{n-1}] \mapsto [\mathsf{I}_0, ..., \mathsf{I}_{n-1}]$ **then**
8: Remove I_i from $\mathbf{I}$
9: **return** $\mathbf{I}$

Table 3: An instance of the table date_dim.

date id	date	year	month	day	quarter	trim-ester
8300	20100830	2010	08	30	3	2
8301	20100931	2010	09	31	3	3
8302	20110105	2011	01	05	1	1
8303	20110106	2011	01	06	1	1
8304	20110401	2011	04	01	2	1

It is straightforward to show the correctness of *Reduce Order OD*. Removing I_i from the list due to the FD $\mathsf{B} \rightarrow \mathsf{I}_i$ is part of the algorithm *Reduce Order* presented in [17]. Assume the order dependency $\mathbf{X} \mapsto \mathbf{Y}$ holds, with $\mathbf{X} = [\mathsf{I}_0, \ldots, \mathsf{I}_{i-1}, \mathsf{I}_{i+1}, \ldots, \mathsf{I}_{n-1}]$ and $\mathbf{Y} = [\mathsf{I}_0, \ldots, \mathsf{I}_{n-1}]$. The interesting order $\mathbf{Y}$ can be replaced by $\mathbf{X}$, as strengthening the order-by conditions is allowed.

Example 2 and Table 3 refer to the *date* domain described in Section 1.

EXAMPLE 2. (ODs over an instance of the date_dim.)
Order dependencies
$[\mathsf{d\_year}, \mathsf{d\_month}, \mathsf{d\_day}] \mapsto [\mathsf{d\_date}]$ *and*
$[\mathsf{d\_date\_id}] \mapsto [\mathsf{d\_year}, \mathsf{d\_month}, \mathsf{d\_day}]$
are satisfied in Table 3. However, order dependencies
$[\mathsf{d\_date\_id}] \mapsto [\mathsf{d\_year}, \mathsf{d\_day}, \mathsf{d\_month}]$ *and*
$[\mathsf{d\_year}, \mathsf{d\_month}] \mapsto [\mathsf{d\_date}]$
are falsified *by Table 3.*

Time and *date* are supported in the SQL standard in a rich manner. The popular TPC-DS benchmark consists of 99 queries. Of these queries, 85 involve date operators and predicates and five involve time operators and predicates. Even if the concept of ODs was only applied to date and time, it could still be of great use for query optimization as shown in Query 1. However, ordered domains are not limited to date and time. They arise in many other domains from business semantics, such as sequence numbers, surrogate keys, measured values such as sales, stock prices and taxes (Example 3).

EXAMPLE 3. (Taxes domain) *Consider the table* `taxes` *which has columns for the taxable* `salary`*, the tax's* `percent` *of the salary,* `taxes` *on the salary, the tax* `group`*, and the tax* `subgroup`*. Let the tax increase with salary and be calculated as a percentage of salary. Tax groups and subgroups create a hierarchy. Employees' tax groups are calculated based*

on their salaries. They increase with salary and take values from A to F. The tax subgroup increases within a group from I to III as the salary goes up. These conditions can be represented as ODs in the following way:

$[\mathsf{salary}] \mapsto [\mathsf{taxes}]$,
$[\mathsf{salary}] \mapsto [\mathsf{percent}]$, *and*
$[\mathsf{salary}] \mapsto [\mathsf{group}, \mathsf{subgroup}]$.

It logically follows from these ODs that

$[\mathsf{salary}] \mapsto [\mathsf{taxes}, \mathsf{group}, \mathsf{subgroup}]$.

This OD can be derived automatically by means of the inference procedure for ODs to be described in Section 5.

Let the table `taxes` have a clustered index on `salary`. A query with `order by taxes, group, subgroup` given the three ODs as declared in Example 3 could then be evaluated using the index on `salary`, as

$[\mathsf{salary}] \mapsto [\mathsf{taxes}, \mathsf{group}, \mathsf{subgroup}]$.

The database administrator could have declared that OD too. However, this is not likely to occur.

We are interested in ODs because `asc` and `desc` bidirectional lexicographical orders are part of SQL standard. Consider a modification of Query 1 with `order by year asc, quarter asc, month asc, day asc, sum(S.sales) desc`. A query plan could then also eliminate `quarter` from the order-by clause as

$$[\overrightarrow{\mathsf{year}}, \overrightarrow{\mathsf{quarter}}, \overrightarrow{\mathsf{month}}, \overrightarrow{\mathsf{day}}, \overleftarrow{\mathsf{sum(sales)}}] \mapsto [\overrightarrow{\mathsf{year}}, \overrightarrow{\mathsf{month}}, \overrightarrow{\mathsf{day}}, \overleftarrow{\mathsf{sum(sales)}}]$$

is satisfied.

Consider again Example 3. Instead of increasing, however, it may happen that tax groups decrease with salary. (This depends on the accounting rules.) This can be expressed as an OD, as marked attributes enable it:

$$[\overrightarrow{\mathsf{salary}}] \mapsto [\overleftarrow{\mathsf{group}}, \overrightarrow{\mathsf{subgroup}}].$$

By the Union inference rule, one can also infer

$$[\overrightarrow{\mathsf{salary}}] \mapsto [\overrightarrow{\mathsf{taxes}}, \overleftarrow{\mathsf{group}}, \overrightarrow{\mathsf{subgroup}}].$$

This OD, therefore, can also be used in query optimization.

3. DECIDABILITY

A goal in any dependency theory is to develop algorithms for testing logical implication, that is, testing whether a dependency is satisfied based on a given set of dependencies. We show how to test logical implication for ODs with a *chase* procedure. [6] Chase is a fixpoint procedure for testing satisfaction of data dependencies in a database [1]. It is used to reason about the consistency and correctness of data design and in query optimization. By providing a complete chase procedure for ODs, we establish that the inference problem for ODs is decidable. We prove that testing logical implication for ODs and inclusion dependencies (IND) is undecidable, however.

3.1 Chasing ODs

We establish a sound and complete chase procedure for ODs for *testing logical implication*.

Definition 6. ($\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$) The problem of *testing logical implication* for ODs is, given a set of ODs $\mathcal{M}$ and an OD $\mathbf{X} \mapsto \mathbf{Y}$, to decide whether $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$.

[6] In preliminary work [18], we focused on *fixing* the table templates (Definition 9), whereas here our technique is based on *detecting* the table templates which falsify the set of ODs $\mathcal{M}$. Therefore, this revised chase procedure is simpler and more efficient. This also means the proof given here can be much more concise.

Table 4: Table representing the split and the swap.

$\mathcal{NV}$	A	B	$\mathcal{MW}$
0...0	0	1	...
0...0	1	0	...

Table 5: Table template.

#	X_1	...	X_k	$\boldsymbol{R} - \{\mathsf{X}_1,...,\mathsf{X}_k\}$		
s	b_1	...	b_k	p_{k+1}	...	p_n
t	b_1	...	b_k	q_{k+1}	...	q_n

(a) Template $\mathbf{r}_0$.

#	X_1	...	X_{j-1}	X_j	$\boldsymbol{R} - \{\mathsf{X}_1,...,\mathsf{X}_j\}$		
s	b_1	...	b_{j-1}	b_j	p_{j+1}	...	p_n
t	b_1	...	b_{j-1}	t_j	q_{j+1}	...	q_n

(b) Template $\mathbf{r}_j$.

An order dependency $\mathbf{X} \mapsto \mathbf{Y}$ can be falsified in two ways, as shown by Theorem 4. We name these two ways *split* and *swap*.

THEOREM 4. *(order dependency)* $\mathbf{X} \mapsto \mathbf{Y}$ *holds iff* $\mathbf{X} \mapsto \mathbf{XY}$ *and* $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

Proof

IF: Suppose $\mathbf{X} \mapsto \mathbf{Y}$. By the Suffix rule $\mathbf{X} \leftrightarrow \mathbf{YX}$. By Prefix and Normalization $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

ONLY IF: Assume $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$ are true. By Transitivity, $\mathbf{X} \mapsto \mathbf{YX}$. By Reflexivity and Transitivity, $\mathbf{X} \mapsto \mathbf{Y}$. □

Definition 7. (split) A *split* with respect to an OD $\mathbf{X} \mapsto \mathbf{XY}$ is a pair of tuples s and t such that $s_{\mathcal{X}} = t_{\mathcal{X}}$ but $s_{\mathcal{Y}} \neq t_{\mathcal{Y}}$. This says that $\mathcal{X}$ does not functionally determine $\mathcal{Y}$.

Definition 8. (swap) A *swap* with respect to an OD $\mathbf{XY} \leftrightarrow \mathbf{YX}$ is a pair of tuples s and t such that: $s \prec_{\mathbf{X}} t$, but $t \prec_{\mathbf{Y}} s$; i.e., s comes before t in any stream satisfying `order by` $\mathbf{X}$, but t comes before s in any stream satisfying `order by` $\mathbf{Y}$. Thus, the swap falsifies $\mathbf{X} \sim \mathbf{Y}$. (Consequently, $\mathbf{X} \mapsto \mathbf{Y}$ is falsified, too.)

EXAMPLE 4. (split and swap) *Let* $\mathbf{X} = \mathbf{NAM}$ *and* $\mathbf{Y} = \mathbf{VBW}$. *Then, there is a split in Table 4 with respect to an OD* $\mathbf{NV} \mapsto \mathbf{NV}\mathsf{AB}$ *and a swap in Table 4 with respect to an OD* $\mathbf{X} \sim \mathbf{Y}$.

We define a *table template* over variables with respect to a given OD. We use table templates to enumerate through all the possible cases where an OD can be falsified by splits and swaps.

Definition 9. (table template) Let $\mathbf{R}$ be a relation schema with n attributes and m be an OD $\mathbf{X} \mapsto \mathbf{Y}$, where $\mathbf{X} = [\mathsf{X}_1, ..., \mathsf{X}_k]$. A table template for an OD m, denoted as $\mathbf{r}_m$, is a table consisting of two tuples s and t, such that it is either $\mathbf{r}_0$ (Table 5a) or $\mathbf{r}_j$ (Table 5b), for j in $[1, ..., k]$. In $\mathbf{r}_0$ and $\mathbf{r}_j$, symbols p_i and q_i represent one of the following three cases, where the *ordering* of variables b_i and t_i is defined as $b_i < t_i$:

1. $p_i = b_i$ and $q_i = b_i$;
2. $p_i = b_i$ and $q_i = t_i$; and
3. $p_i = t_i$ and $q_i = b_i$;

Example 5 presents how to apply a *mapping* (Definition 10) to a table template.

Definition 10. (mapping $\mathbf{r}_m$ to $\varphi(\mathbf{r}_m)$) Let $\mathbf{r}_m$ be a table template from Definition 9. A mapping of $\mathbf{r}_m$ to $\varphi(\mathbf{r}_m)$

Table 6: Mapping.

#	A	B	C
s	b_1	b_3	t_4
t	b_1	t_3	b_4

(a) $\mathbf{r}_m$.

#	A	B	C
s	5	0	8
t	5	1	7

(b) Instance $\varphi(\mathbf{r}_m)$.

is *any* instance with values that satisfy the *ordering* from Definition 9.

EXAMPLE 5. *Consider Table 6a and 6b as one of the possible mappings from Definition 10. In fact, it can be any relation instance which satisfies the Definition 9 for* ordering *of the variables. (The* ordering *of variables b_i and t_i is defined as $b_i < t_i$)*

LEMMA 1. *Let $\mathbf{r}_m$ be a table template (Definition 9) and $\varphi(\mathbf{r}_m)$ be a mapping from $\mathbf{r}_m$ (Definition 10). Then $\mathbf{r}_m \models \mathbf{X} \mapsto \mathbf{Y}$ iff $\varphi(\mathbf{r}_m) \models \mathbf{X} \mapsto \mathbf{Y}$.*

Proof

By Definition 10 ordering of values in $\varphi(\mathbf{r}_m)$ corresponds to the ordering of variables in $\mathbf{r}_m$, respectively. □

Definition 11. (tableaux $\mathbf{T}_m$) Let m be an OD $\mathbf{X} \mapsto \mathbf{Y}$. We define $\mathbf{T}_m$ to be the set of all table templates $\mathbf{r}_m$, as we defined in Definition 9.

Note that $\mathbf{T}_m$ is a *set* of table templates, each consisting of two rows. The *chase* of $\mathbf{T}_m$ is defined as follows.

Definition 12. (chase of tableaux $\mathbf{T}_m$) The chase of $\mathbf{T}_m$ over a set of ODs $\mathcal{M}$ denoted as $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}}$ is defined by $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}} = \{\mathbf{r}_m \mid \mathbf{r}_m \in \mathbf{T}_m \wedge \mathbf{r}_m \models \mathcal{M}\}$. Furthermore, $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}}$ satisfies $\mathbf{X} \mapsto \mathbf{Y}$, denoted by $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$, *iff*, for all $\mathbf{r}_m \in \text{CHASE}_{\mathbf{T}_m,\mathcal{M}}$, $\mathbf{r}_m \models \mathbf{X} \mapsto \mathbf{Y}$. $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}}$ satisfies the set of ODs $\mathcal{M}'$, which is denoted as $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}} \models \mathcal{M}'$, *iff*, for all $\mathbf{X} \mapsto \mathbf{Y} \in \mathcal{M}'$, $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$.

THEOREM 5. *(chase procedure for ODs is sound and complete) Let $\mathcal{M}$ be a set of ODs over $\mathbf{R}$ and m be an OD $\mathbf{X} \mapsto \mathbf{Y}$. Then $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ iff $CHASE_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$.*

Proof

IF: Assume $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}} \not\models \mathbf{X} \mapsto \mathbf{Y}$. By Definition 12, there exists $\mathbf{r}_m \in \text{CHASE}_{\mathbf{T}_m,\mathcal{M}}$ such that $\mathbf{r}_m \not\models \mathbf{X} \mapsto \mathbf{Y}$. By Definition, 12 $\mathbf{r}_m \models \mathcal{M}$. Hence, there is a mapping φ to generate a relation instance $\varphi(\mathbf{r}_m)$. By Lemma 1, $\varphi(\mathbf{r}_m) \models \mathcal{M}$, but in addition $\varphi(\mathbf{r}_m) \not\models \mathbf{X} \mapsto \mathbf{Y}$. We have found a relation instance which satisfies $\mathcal{M}$ but does not satisfy $\mathbf{X} \mapsto \mathbf{Y}$, which implies that $\mathcal{M} \not\models \mathbf{X} \mapsto \mathbf{Y}$.

ONLY IF: Assume $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$. Let s and t be *any* two tuples in *any* relation $\mathbf{r}$ such that $s \preceq_{\mathbf{X}} t$ and that satisfies the set of ODs $\mathcal{M}$. We would like to present that $s \preceq_{\mathbf{Y}} t$. Let $\mathbf{r}_m \in \mathbf{T}_m$. Let $\mathbf{r}_m = \{p, q\}$ be the template relation such that $\varphi(p) = s$ and $\varphi(q) = t$. It is possible always to find such a pair of tuples $\mathbf{r}_m$ since $\mathbf{T}_m$ considers all possibilities of two tuples which satisfy condition $s \preceq_{\mathbf{X}} t$. Therefore, we have $\varphi(\mathbf{r}_m) = \{s, t\}$ and $\varphi(\mathbf{r}_m) \models \mathcal{M}$. By Lemma 1, it follows that $\mathbf{r}_m \models \mathcal{M}$. It follows by Definition 12 that $\mathbf{r}_m \in \text{CHASE}_{\mathbf{T}_m,\mathcal{M}}$. Since we assumed that $\text{CHASE}_{\mathbf{T}_m,\mathcal{M}} \models \mathbf{X} \mapsto \mathbf{Y}$, we have $\mathbf{r}_m \models \mathbf{X} \mapsto \mathbf{Y}$. This implies that $\varphi(\mathbf{r}_m) \models \mathbf{X} \mapsto \mathbf{Y}$ by Lemma 1. Hence, $s \preceq_{\mathbf{Y}} t$. □

THEOREM 6. *(decidable) The implication problem of ODs is decidable.*

Proof

By Theorem 5, testing logical implication problem of ODs is decidable as the chase procedure is a sound and complete inference algorithm for ODs. □

THEOREM 7. *(complexity of chase) The complexity of building templates for the ODs chase procedure is exponential.*

Proof

By Definition 9 there are 3^{n-k} templates for $\mathbf{r}_0$ and 3^{n-j} templates for each $\mathbf{r}_j$. Therefore, there are $(3^n + 3^{n-k})/2$ templates in total by geometric progression: $O(3^n)$. □

3.2 ODs with inclusion dependencies

There is a strong relationship between ODs and FDs. Any OD implies an FD, modulo lists and sets, but not vice versa.

LEMMA 2. *(relationship between ODs and FDs) For every instance $\mathbf{r}$ of relation $\mathbf{R}$, if an OD $\mathbf{X} \mapsto \mathbf{Y}$ holds, then the FD $\mathcal{X} \rightarrow \mathcal{Y}$ is true.*

Proof

Let rows s and $t \in \mathbf{r}$. Assume that $s_{\mathcal{X}} = t_{\mathcal{X}}$. Hence, $s \preceq_{\mathbf{X}} t$ and $t \preceq_{\mathbf{X}} s$. By definition of an OD, $s \preceq_{\mathbf{Y}} t$ and $t \preceq_{\mathbf{Y}} s$. Therefore, $s_{\mathcal{Y}} = t_{\mathcal{Y}}$ holds. □

Furthermore, there exists a correspondence between FDs and ODs.

THEOREM 8. *(correspondence between ODs and FDs) For relation $\mathbf{R}$, for every instance $\mathbf{r}$ of it, $\mathcal{X} \rightarrow \mathcal{Y}$ iff $\mathbf{X} \mapsto \mathbf{XY}$, for any list $\mathbf{X}$ that order the attributes of $\mathcal{X}$and any list $\mathbf{Y}$ likewise for $\mathcal{Y}$.*

Proof

IF: Assume an OD $\mathbf{X} \mapsto \mathbf{XY}$ does not hold. This means, there exists s and $t \in \mathbf{r}$, such that $s \preceq_{\mathbf{X}} t$ but $s \not\preceq_{\mathbf{XY}} t$ by Definition 5. Therefore, $s_{\mathcal{X}} = t_{\mathcal{X}}$ and $s \prec_{\mathbf{Y}} t$. Also $s \prec_{\mathbf{Y}} t$ implies that $s_{\mathcal{Y}} \neq t_{\mathcal{Y}}$. Therefore, $\mathcal{X} \rightarrow \mathcal{Y}$ is not satisfied.

ONLY IF: By Lemma 2 if $\mathbf{X} \mapsto \mathbf{XY}$, then $\mathcal{X} \rightarrow \mathcal{XY}$. The FD $\mathcal{XY} \rightarrow \mathcal{Y}$ holds by Armstrong axiom Reflexivity [2]. Hence by Armstrong axiom Transitivity, $\mathcal{X} \rightarrow \mathcal{Y}$. □

The inference problem for ODs with INDs is undecidable.

THEOREM 9. *(Undecidable for ODs with INDs)*

Testing logical implication for ODs with INDs is undecidable.

Proof

Testing implication for FDs with INDs is known to be undecidable [1]. Therefore, testing logical implication for ODs with INDs is undecidable, too as there is a correspondence between ODs and FDs (Theorem 8). □

4. COMPLEXITY

We show that the inference problem for ODs is co-NP-complete. More specifically, we show that inference problem for UODs and the inference problem of FDs from ODs are co-NP-complete. FD inference from UODs, a restricted case, is polynomially decidable, however.

4.1 OD Inference

We introduce first the notation which permits us to translate instances of 3-SAT into instances of the decision problem for testing logical implication for ODs. We assume the reader is familiar with NP-completeness in general, with the 3-SAT problem, and with reducibility [8].

Definition 13. Let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a set of propositional variables for an arbitrary finite n, and let $\overline{\mathcal{P}} = \{\neg p_1, \ldots, \neg p_n\}$. Let $\mathcal{F}$ be a formula written over the propositional variables in $\mathcal{P}$ and their negations in conjunctive normal form with k clauses, each a disjunction of length three, for an arbitrary finite k.

For $i \in \{1,\ldots,k\}$, let $\mathsf{V}_{i,1} \vee \mathsf{V}_{i,2} \vee \mathsf{V}_{i,3}$ represent clause i such that

$\mathsf{V}_{i,1} \in (\mathcal{P} \cup \overline{\mathcal{P}})$,
$\mathsf{V}_{i,2} \in (\mathcal{P} \cup \overline{\mathcal{P}}) - \{\mathsf{V}_{i,1}\}$, and
$\mathsf{V}_{i,3} \in (\mathcal{P} \cup \overline{\mathcal{P}}) - \{\mathsf{V}_{i,1}, \mathsf{V}_{i,2}\}$,

without loss of generality.

Call any such $\mathcal{F}$ a *3-SAT candidate*. Call any such *3-SAT candidate* $\mathcal{F}$ for which there exists a truth assignment over $\mathcal{F}$'s $\mathcal{P}$ which satisfies $\mathcal{F}$ a *3-SAT instance*.

3-SAT is the collection of 3-SAT instances.

LEMMA 3. [8] **3-SAT** *is in NP-complete.*

Definition 14. Let $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ be an arbitrary pair of a finite set $\mathcal{M}$ of UODs and a *target* UOD $\mathbf{X} \mapsto \mathbf{Y}$ constructed over the attributes that appear in $\mathcal{M}$.

Call any such $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ an *UODI candidate*. Call any such $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ for which $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ an *UODI instance*.

UODI is the collection of UODI instances. This is the set-theoretic characterization of *the inference decision problem for UODs.*

THEOREM 10. *The inference decision problem for unidirectional order dependencies is coNP-complete. Therefore,* **UODI** *is in coNP-complete.*

Proof

Reduction from 3-SAT.

Given a 3-SAT candidate $\mathcal{F}$ (Def. 13), we construct an UODI candidate $\langle \mathcal{M}_\mathcal{F}, [\mathsf{T}] \sim [\mathsf{F}] \rangle$ (Def. 14).

Construction.

$\mathcal{M}_\mathcal{F}$ is constructed as follows. For each p_i, $i \in \{1,\ldots,n\}$, from $\mathcal{F}$, we introduce four attributes to appear in $\mathcal{M}_\mathcal{F}$: $\mathsf{P}_{i,t}$, $\mathsf{P}_{i,f}$, $\mathsf{Q}_{i,t}$, and $\mathsf{Q}_{i,f}$. (Our intent is that $[\mathsf{P}_{i,t}, \mathsf{P}_{i,f}]$ will mirror the truth value of p_i from $\mathcal{F}$ in a given truth assigment, and $[\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}]$ will mirror the truth value of $\neg p_i$ in that truth assigment.)

For $i \in \{1,\ldots,n\}$, add the following order dependencies for $\mathsf{P}_{i,t}$ and $\mathsf{P}_{i,f}$ to $\mathcal{M}_\mathcal{F}$:

1. $[\mathsf{P}_{i,t}] \sim [\mathsf{T}]$
2. $[\mathsf{P}_{i,f}] \sim [\mathsf{F}]$
3. $[\mathsf{P}_{i,t}] \sim [\mathsf{P}_{i,f}]$
4. $[\mathsf{P}_{i,t}, \mathsf{P}_{i,f}, \mathsf{T}] \sim [\mathsf{P}_{i,t}, \mathsf{P}_{i,f}, \mathsf{F}]$

Likewise, for $i \in \{1,\ldots,n\}$, symmetrically add the "same" order dependencies for $\mathsf{Q}_{i,t}$ and $\mathsf{Q}_{i,f}$ to $\mathcal{M}_\mathcal{F}$:

5. $[\mathsf{Q}_{i,t}] \sim [\mathsf{T}]$
6. $[\mathsf{Q}_{i,f}] \sim [\mathsf{F}]$
7. $[\mathsf{Q}_{i,t}] \sim [\mathsf{Q}_{i,f}]$
8. $[\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}, \mathsf{T}] \sim [\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}, \mathsf{F}]$

For $i \in \{1,\ldots,n\}$, add to $\mathcal{M}_\mathcal{F}$:

9. $[\mathsf{P}_{i,t}, \mathsf{Q}_{i,t}, \mathsf{T}] \sim [\mathsf{P}_{i,t}, \mathsf{Q}_{i,t}, \mathsf{F}]$
10. $[\mathsf{P}_{i,f}, \mathsf{Q}_{i,f}, \mathsf{T}] \sim [\mathsf{P}_{i,f}, \mathsf{Q}_{i,f}, \mathsf{F}]$

Next, we encode the clauses. For each clause, $i \in \{1,\ldots,k\}$, from $\mathcal{F}$, we introduce three attributes: $\mathsf{V}_{i,1}$, $\mathsf{V}_{i,2}$, and $\mathsf{V}_{i,3}$. For $i \in \{1,\ldots,k\}$, $j \in \{1,\ldots,3\}$, add one OD to $\mathcal{M}_\mathcal{F}$ as follows. If $\mathsf{V}_{i,j} = p_l$ (for a given $l \in \{1,\ldots,n\}$) in $\mathcal{F}$, add to $\mathcal{M}_\mathcal{F}$:

11. $[\mathsf{V}_{i,j}] \sim [\mathsf{P}_{l,t}, \mathsf{P}_{l,f}]$

Else, $\mathsf{V}_{i,j} = \neg p_l$ (for a given $l \in \{1,\ldots,n\}$) in $\mathcal{F}$; add to $\mathcal{M}_\mathcal{F}$:

12. $[\mathsf{V}_{i,j}] \sim [\mathsf{Q}_{l,t}, \mathsf{Q}_{l,f}]$

Finally, for each clause $i \in \{1,\ldots,k\}$ in $\mathcal{F}$, we introduce an attribute C_i, and we add to $\mathcal{M}_\mathcal{F}$:

13. $[\mathsf{C}_i] \mapsto [\mathsf{T}]$
14. $[\mathsf{C}_i] \mapsto [\mathsf{V}_{i,1}, \mathsf{V}_{i,2}, \mathsf{V}_{i,3}, \mathsf{F}]$

Polynomial reduction.

The translation procedure above of a 3-SAT candidate into an UODI candidate is clearly polynomial in the size of $\mathcal{F}$.

Witness.

We can build a counter-example for a given UODI candidate to demonstrate that it is *not* an UODI instance, in **UODI**.

A pair of tuples is necessary and sufficient to falsify $[\mathsf{T}] \sim [\mathsf{F}]$. Therefore, $\mathcal{M}_\mathcal{F} \not\models [\mathsf{T}] \sim [\mathsf{F}]$ *iff* we can construct a two-tuple table $\mathbf{t}$ over the attributes appearing in $\mathcal{M}_\mathcal{F}$ that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$, but that does not falsify any order dependency in $\mathcal{M}_\mathcal{F}$ (thus *satisfies* $\mathcal{M}_\mathcal{F}$). Between the two tuples in $\mathbf{t}$, T will have different values, F will have different values, and the values of T and F will be anti-monotonic. Let the two values for T and for F in $\mathbf{t}$ be 0 and 1, without loss of generality. We write the tuples in $\mathbf{t}$ in a fixed order in our discussion such that $t_{\mathsf{T},\mathsf{F}} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$, without loss of generality. Conceptually, a transition from 0 to 1, as in $t_\mathsf{T} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$, encodes *true*; a transition from 1 to 0, as in $t_\mathsf{F} = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$, represents *false*.

We can always build a two-tuple table $\mathbf{t}$ such that $t_{\mathsf{T},\mathsf{F}} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ (which is necessary and sufficient to falsify $[\mathsf{T}] \sim [\mathsf{F}]$) which satisfies ODs 1–13 of $\mathcal{M}_\mathcal{F}$. Let us construct such a $\mathbf{t}$. Because of OD 1, $t_{\mathsf{P}_{i,t}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$. (If only a single value appears in $\mathbf{t}$ for an attribute, we can assume that value is 0, without loss of generality.) Because of OD 2, $t_{\mathsf{P}_{i,f}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$. Because of OD 3, $t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f}} \neq \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$. Because of OD 4, $t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f}} \neq \left[\begin{smallmatrix} 0 & 0 \\ 0 & 0 \end{smallmatrix}\right]$. (Otherwise, OD 4 would be falsified by $\mathbf{t}$, since $t_{\mathsf{T},\mathsf{F}} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.) Therefore, $t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f}} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right]$.

From ODs 5–8, it symmetrically follows that $t_{\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right]$.

From ODs 9–10, it further follows that

$$t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{smallmatrix}\right]$$

or

$$t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right].$$

Thus, $\mathbf{t} \models [\mathsf{P}_{i,t}, \mathsf{P}_{i,f}] \not\sim [\mathsf{Q}_{i,t}, \mathsf{Q}_{i,f}]$.

For any $\mathsf{V}_{i,j}$ such that $\mathsf{V}_{i,j} = p_l$ for a given l in $\mathcal{F}$, so OD 11 is in $\mathcal{M}_\mathcal{F}$ for i, we know the following:

- if $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right]$, then $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$;
- else $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$ and $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$.

For any $\mathsf{V}_{i,j}$ such that $\mathsf{V}_{i,j} = \neg p_l$ for a given l in $\mathcal{F}$ *instead*, so OD 12 is in $\mathcal{M}_\mathcal{F}$ for i, we know the following:

- if $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right]$, then $t_{\mathsf{Q}_{l,t},\mathsf{Q}_{l,f}} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$ and $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$;
- else $t_{\mathsf{P}_{l,t},\mathsf{P}_{l,f}} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$, $t_{\mathsf{Q}_{l,t},\mathsf{Q}_{l,f}} = \left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right]$ and $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$.

To satisfy ODs 13, for $i \in \{1,\ldots,k\}$, it must be that $t_{\mathsf{C}_i} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$ since $t_\mathsf{T} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$.

In coNP.

It is not always possible further to set values for the $\mathsf{V}_{i,j}$'s in such a way that $\mathbf{t}$ also satisfies the ODs 14, for $i \in \{1,\ldots,n\}$, $j \in \{1,\ldots,3\}$, and so satisfies $\mathcal{M}_\mathcal{F}$ completely. When we can also set values for the $\mathsf{V}_{i,j}$'s so that $\mathbf{t}$ also satisfies the ODs 14 too, then $\mathbf{t}$ suffices as a *witness* that $\langle \mathcal{M}_\mathcal{F}, [\mathsf{T}] \sim [\mathsf{F}] \rangle \notin$ **UODI**.

Correspondence.

$\mathcal{F} \in$ **3-SAT** *iff* $\langle \mathcal{M}_\mathcal{F}, [\mathsf{T}] \sim [\mathsf{F}] \rangle \notin$ **UODI**.

Consider two-tuple tables $\mathbf{t}$ that satisfy the ODs 1–10 and 13 from $\mathcal{M}_\mathcal{F}$, but that falsify $[\mathsf{T}] \sim [\mathsf{F}]$. There is a one-to-one mapping between truth assigments over the p_i, for $i \in \{1,\ldots,n\}$, in $\mathcal{F}$ and settings for $\mathsf{P}_{i,t}$ in such $\mathbf{t}$. For $i \in \{1,\ldots,n\}$, if $p_i = \mathit{true}$ in the truth assignment, set

$$t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{smallmatrix}\right];$$

else ($p_i = \mathit{false}$), set

$$t_{\mathsf{P}_{i,t},\mathsf{P}_{i,f},\mathsf{Q}_{i,t},\mathsf{Q}_{i,f}} = \left[\begin{smallmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right].$$

IF: There is some truth assigment over $p_1, \ldots, p_n$ that satisfies $\mathcal{F}$.

We construct a two-tuple table $\mathbf{t}$ based on this truth assignment that satisfies $\mathcal{M}_{\mathcal{F}}$ for ODs 1–13, and that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$, as above (in the *Witness* part). For $i \in \{1, \ldots, n\}$, assign values for $\mathsf{P}_{i,t}$, $\mathsf{P}_{i,f}$, $\mathsf{Q}_{i,t}$, and $\mathsf{Q}_{i,f}$ according to the truth assignment mapping above.

To satisfy further ODs 14, we must be able to assign values to the $\mathsf{V}_{i,j}$'s that suffice. For $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, 3\}$, if $V_{i,j} = \mathit{true}$, set $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$. This satisfies the OD 11 or 12 added to $\mathcal{M}_{\mathcal{F}}$ for i, given how we assigned $\mathsf{P}_{i,t}$, $\mathsf{P}_{i,f}$, $\mathsf{Q}_{i,t}$, and $\mathsf{Q}_{i,f}$ based on p_i's truth value. Otherwise ($V_{i,j} = \mathit{false}$), set $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$. This satisfies either the OD 11 or 12 for i, j, vacuously.

Since, for each $i \in \{1, \ldots, k\}$, at least one of $V_{i,1}$, $V_{i,2}$, and $V_{i,3}$ is *true* in the truth assignment, at least one of $t_{\mathsf{V}_{i,1}}$, $t_{\mathsf{V}_{i,2}}$, or $t_{\mathsf{V}_{i,3}}$is $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$. Thus, $\mathbf{t}$ as constructed satisfies ODs 1−14, and so all of $\mathcal{M}_{\mathcal{F}}$.

ONLY IF: There is no truth assignment that satisfies $\mathcal{F}$.

For any arbitrary truth assignment, we can build a two-tuple table $\mathbf{t}$ that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$ based on the truth assignment mapping that satisfies ODs 1–13, as done in the *if* part. We next try to assign values to the $\mathsf{V}_{i,j}$'s in such a way that $\mathbf{t}$ satisfies ODs 14.

Since the truth assignment does not satisfy $\mathcal{F}$, there is some clause i such that $V_{i,1}$, $V_{i,2}$, and $V_{i,3}$ are each *false*. The OD 14 for i will be falsified. For each $V_{i,j}$, as either the OD 11 or 12 is satisfied accordingly, $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$ *or* $\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$.

If, for any $V_{i,j}$, $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$, OD 14 is falsified since $t_{\mathsf{C}_i} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$. If instead, for all $V_{i,j}$, $t_{\mathsf{V}_{i,j}} = \left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right]$, OD 14 is still falsified, since $t_{\mathsf{F}} = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$.

No two-tuple table $\mathbf{t}$ that falsifies $[\mathsf{T}] \sim [\mathsf{F}]$ can be constructed that satisfies $\mathcal{M}_{\mathcal{F}}$. Any table $\mathbf{t}$ therefore either satisfies $[\mathsf{T}] \sim [\mathsf{F}]$ or falsifies $\mathcal{M}_{\mathcal{F}}$. □

4.2 FD inference over ODs

Definition 15. Let $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ be an arbitrary pair of a finite set $\mathcal{M}$ of ODs and an *target* UOD $\mathbf{X} \mapsto \mathbf{Y}$ constructed over the attributes that appear in $\mathcal{M}$.

Call any such $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ an *ODI candidate*. Call any such $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{Y} \rangle$ for which $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ an *ODI instance*.

ODI is the collection of ODI instances. This is the set-theoretic characterization of *the inference decision problem for ODs*.

COROLLARY 11. *The inference decision problem for order dependencies is coNP-complete. That is,* **ODI** *is in coNP-complete.*

Proof

Follows directly from Theorem 10 as **UODI** is a proper sub-problem of **ODI** (Definition 15). □

Functional-dependency inference for UODs is polynomial. In fact, it can be done in linear time (Theorem 12). This does not contradict Theorem 10; the type of inference that is *hard* for unidirectional ODs is order compatiblity, $\mathbf{X} \sim \mathbf{Y}$ (as employed in Proof 10).

Let the length of the representation of $\mathcal{M}$, the string of concatenated left-hand and right-hand sides of ODs, be denoted by $|\mathcal{M}|$.

THEOREM 12. *(testing logical implication $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$) Testing, whether $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$ ($\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$) can be accomplished in $O(|\mathcal{M}|)$ time. (This includes finding the closure for FDs, $\mathcal{X}^+$.)*

Proof

Assume $\mathcal{M}' = \{\mathbf{X} \mapsto \mathbf{XY}, \mathbf{XY} \leftrightarrow \mathbf{YX} \mid \mathbf{X} \mapsto \mathbf{Y} \in \mathcal{M}\}$. In [19], we have shown that $\mathcal{F} = \{\mathcal{X} \rightarrow \mathcal{Y} \mid \mathbf{X} \mapsto \mathbf{Y} \in \mathcal{M}'\}$ is a set of FDs which enables to compute closure for FDs $\mathcal{X}^+$ over the set of UODs $\mathcal{M}$. Therefore, as testing logical implication of a FD $\mathcal{X} \rightarrow \mathcal{Y}$ has already been shown to be linear in [5]. This implies that testing $\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$ can be also accomplished in $O(|\mathcal{M}|)$. The same applies to $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$. □

This is not the same case, however, for general order dependencies. Both the inference problems for functional dependencies (embedded within the ODs), $\mathbf{X} \mapsto \mathbf{XY}$, and for order compatibility, $\mathbf{X} \sim \mathbf{Y}$, are *hard*.

We call an attribute a *constant* if, for any table that satisfies the set of ODs $\mathcal{M}$, it can have only a single value occurring in the table.

Definition 16. (constant) A marked attribute A is called a *constant* with respect to $\mathcal{M}$ *iff* $\mathcal{M} \models [\,] \mapsto \mathsf{A}$.

Interestingly, an order specification may become empty. We are able to express this with order dependencies. For instance, if there is a predicate in the `where` clause $\mathsf{A} = 150$, which means that A is a *constant* in the scope of the query, this predicate yields $[\,] \mapsto \mathsf{A}$. Given $\mathbf{I} = [\mathsf{A}]$, note that $\mathbf{I}$ may reduce to empty.

THEOREM 13. *Functional-dependency inference for order dependencies is co-NP-complete.*

Proof

It suffices to show that any 3-SAT candidate (Definition 13) can be reduced to an ODI candidate of the form $\langle \mathcal{M}, \mathbf{X} \mapsto \mathbf{XY} \rangle$ (Definition 15). We reduce any 3-SAT candidate to an ODI candidate of the form $\langle \mathcal{M}, [\,] \mapsto [\overrightarrow{\mathsf{T}}] \rangle$. [7]

Construct an ODI candidate from a given 3-SAT candidate in the very way the UODI candidate—also an ODI candidate—is constructed in Proof 10. (Recall every unmarked attribute in ODs 1–14 is ascending, by default.) Add one more OD to $\mathcal{M}$:

15. $[\overrightarrow{\mathsf{F}}] \mapsto [\overleftarrow{\mathsf{T}}]$.

A two-tuple table $\mathbf{t}$ is a necessary and sufficient witness that $\mathcal{M} \not\models [\,] \mapsto [\overrightarrow{\mathsf{T}}]$. Let $t_{\mathsf{T}} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$, without loss of generality. Then, $t_{\mathsf{F}} = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$, given $\mathbf{t}$ satisfies OD 15.

The rest of the proof then proceeds the same as for Theorem 10. □

5. INFERENCE PROCEDURE

We introduce an inference procedure for testing logical implication for a *restricted domain* for UODs. The additional order property to be guaranteeed over the schema is intuitive, holds for all real-world business domains that we have encountered, and can easily be verified whether it holds. Thus, it is a natural restriction on the domain. We develop an inference procedure for UODs which is sound and complete when applied over a database that satisfies the property. Our inference procedure is efficient, with a reasonable polynomial complexity bound with respect to schema complexity.

[7] A simpler reduction from 3-SAT to ODI is possible which suffices. A single pair of attributes can be forced to be anti-monotonic: $[\overrightarrow{\mathsf{P}}] \leftrightarrow [\overleftarrow{\mathsf{Q}}]$, just as with T and F. Then, P_i and Q_i can be used to encode the truth assignment of p_i. Given that we preceded with Theorem 10, however, the proof given here is more concise.

Table 7: Showing Lack of Transitivity.

A	B	C
0	0	1
1	0	0
2	2	2

5.1 A Natural Domain

In [14], an axiomatization for UODs (defined as we have in this paper) over a restricted domain is presented. The authors call these *temporal functional dependencies* (TFDs), focusing on the time domain, and they provide axiomatization for TFDs. A TFD $\mathbf{X} \rightarrow \mathcal{Y}$ means that $\forall\, \mathsf{A} \in \mathcal{Y}.\ \mathbf{X} \mapsto [\mathsf{A}]$, in which $\mathbf{X}$ describes time and the attributes in $\mathcal{Y}$ are time variants. The domain is too restricted, unfortunately, to be of use to us. It effectively restricts one to ODs of the form with just a single attribute on the right-hand side (e.g., $\mathbf{X} \mapsto [\mathsf{A}]$). In many of our examples, in particular, in Examples 2 and 3, we need UODs with lists of multiple attributes on the right-hand side. Thus, TFDs do not suffice. Furthermore, no inference procedure for TFDs was defined.

We can take the same tactic to find a natural property by which we can restrict our database domains to make the inference problem tractable, but still cover real-world domains. Our axiomatization in Section 2 yields us insight into how this can be done.

It is surprising that the *order-compatibility* relation '$\sim$' (Definition 4) is *not* transitive as shown in Example 6. (By *Transitivity* axiom (Theorem 1) the *order* relation ('$\mapsto$') is.)

Example 6. (Order compatibility is not transitive.) *Assume* $\mathcal{M} = \{\mathsf{A} \sim \mathsf{B}, \mathsf{B} \sim \mathsf{C}\}$. *The Table 7 satisfies the set of UODs* $\mathcal{M}$. *However, it falsifies* $\mathsf{A} \sim \mathsf{C}$. *This demonstrates that the order-compatibility relation is* not *transitive.*

If we restrict our domains to have a property that guarantees a limited form of transitivity over *order-compatibility*, then we can make an efficient inference procedure for UODs.[8] The property we prescribe is *transitivity of order compatibility over single attributes.* Let us call a domain a *transitive* domain if it satisfies this property.

Definition 17. (transitivity of order compatibility) A domain (relation schema) has the property of *transitivity of order compatibility iff* it can be quaranteed that, for each relation $\mathbf{R}$ in the schema, for any three attributes A, B, and C, where B is not a constant, if $\mathsf{A} \sim \mathsf{B}$ and $\mathsf{B} \sim \mathsf{C}$, then $\mathsf{A} \sim \mathsf{C}$. If a domain has this property, we call it a *transitive* domain.

Example 7. (Transitivity over order compatibility.)
Order compatibilities

$\mathsf{quarter} \sim \mathsf{month}$ *and*
$\mathsf{month} \sim \mathsf{trimester}$

are satisfied in Table 3 by Date domain. Also, so is

$\mathsf{quarter} \sim \mathsf{trimester}$

Hence, the transitivity property holds over order compatibility.

All of the real-world business domains we have explored including the TPC-DS schema, and the examples which are used in this paper (the Date and Taxes domain) are *transitive*, by Definition 17. One can argue that breaking the underlying property in data can be only done by contrivance. Thus, this restriction is quite *natural*, as it covers the cases one sees in practice. Domains can be tested if they are transitive in a straightforward way, by enumeration.

[8]It is this lack of transitivity over the order-compatible relation generally that is at the heart of the high complexity for the inference problem over general domains.

5.2 An Inference Procedure for UODs

Let $\mathcal{M} = \{m_0, ..., m_{n-1}\}$ be a set of UODs defined over the set of attributes $\mathcal{U} = \{\mathsf{A}_0, ..., \mathsf{A}_{m-1}\}$. The set $\mathcal{M}$ is represented as a string of pairs, each pair representing an UOD (the left-hand and right-hand sides of the dependency). Each side is a list of attributes. Let the length of the representation of $\mathcal{M}$, the string of concatenated left-hand and right-hand sides, be denoted by $|\mathcal{M}|$. Also, let $(|m| = |\mathbf{X}| + |\mathbf{Y}|)$. Since each attribute in m appears in at least one UOD of $\mathcal{M}$, we can assume that $|m| \leq |\mathcal{M}|$.

Let m be the UOD $\mathbf{X} \mapsto \mathbf{Y}$, for which both $\mathbf{X}$ and $\mathbf{Y}$ are defined over the set of attributes $\mathcal{U}$. We denote the length of $\mathbf{X}$ and $\mathbf{Y}$ as $|\mathbf{X}|$ and $|\mathbf{Y}|$, respectively.

We first present the important elements of the algorithm for testing logical implication for natural domains of UODs. Lastly, we establish that the algorithm is sound and complete in Theorem 14.

The algorithm *TestOrderDependency* (Algorithm 2) tests logical implication for transitive domains of UODs. It invokes algorithms *TestFunctionalDependency* and *TestOrderCompatible* (Algorithm 3). Algorithm *TestFunctionalDependency* performs a test if $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$ which by Theorem 8 implies an FD, $\mathcal{M} \models \mathcal{X} \rightarrow \mathcal{Y}$. Algorithm *TestOrderCompatible* tests if $\mathcal{M} \models \mathbf{XY} \leftrightarrow \mathbf{YX}$ ($\mathcal{M} \models \mathbf{X} \sim \mathbf{Y}$). These parts combine to complete the proof of soundness and completeness of our inference procedure for UODs over transitive domains. Since by Theorem 4 $\mathbf{X} \mapsto \mathbf{Y}$ holds *iff* $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

Algorithm 2 TestOrderDependecy

Input: A set $\mathcal{M}$ of n unidirectional order dependencies on attributes $\{\mathsf{A}_0, ..., \mathsf{A}_{m-1}\}$ and an UOD $\mathbf{X} \mapsto \mathbf{Y}$.
Output: "true" if $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$; "false" otherwise.
Global data structures:

a. Attributes are represented by integers between 0 and m-1.
b. UODs in $\mathcal{M}$ are represented by integers between 0 and n-1.
c. LS[0:$n-1$], RS[0:$n-1$] are arrays of lists, containing the attributes in the left and right side of each UOD.
d. DEPEND[0:$m-1$] is an array of attributes found to be *functionally dependent* on given set of attributes.
e. OC[0:$n-1$; 0:1] is a two dimensional array of *order compatible* dependencies with *single attribute* on the left and right side.
f. LX and LY are lists of attributes represented by integers, corresponding to $\mathbf{X}$ and $\mathbf{Y}$ respectively.

1: DEPEND ← *TestFunctionalDependency*($\mathcal{M}$, $\mathcal{X}$)
2: **if** exists i in LY such that DEPEND[i] = "false" **then**
3: *result* ← "false"
4: **return** *result*
5: **else**
6: *result* ← *TestOrderCompatible*
7: **return** *result*

Theorem 12 states that testing whether $\mathbf{X} \mapsto \mathbf{XY}$, which corresponds to an FD $\mathcal{X} \rightarrow \mathcal{Y}$ (Theorem 8), can be achieved in linear time. Notice that we assume there is a *TestFunc-*

tionalDependency algorithm which finds a closure of a given set of attributes $\mathcal{X}$, since the authors of [5] designed a *linear* algorithm for finding closure over FDs.

Testing if $\mathbf{X} \sim \mathbf{Y}$ is more involved and complex. We observe that $\mathcal{M} \not\models \mathbf{X} \sim \mathbf{Y}$ *iff* we are able to construct a table $\mathbf{t}$ that *satisfies* set of UODs $\mathcal{M}$ and consists of two rows which have a *swap* (see Definition 8 and Example 4 in Section 3.1) with respect to $\mathbf{X}$ and $\mathbf{Y}$. In the table $\mathbf{t}$ that we construct, we shall use integer values for the *cells* without lost of generality. (A cell is a given column entry of a given row.)

We test if $\mathbf{X} \sim \mathbf{Y}$ in the algorithm *TestOrderCompatible* (Algorithm 3). For each pair of attributes A in $\mathbf{X}$ and B in $\mathbf{Y}$, we test in an algorithm *TestSingleOrderCompatible* (Algorithm 4) whether we can construct a table $\mathbf{t}$ described above with a swap with respect to $\mathsf{A} \sim \mathsf{B}$ with attributes prefixing A and B, in lists $\mathbf{X}$ and $\mathbf{Y}$, respectively, being *constants* (Definition 16) within table $\mathbf{t}$, such that table $\mathbf{t}$ satisfies the set of ODs $\mathcal{M}'$. (Let $\mathbf{P}$ be a concatenated attributes prefixing A and B. We consider $\mathcal{M}' = \mathcal{M} \cup \{[\,] \mapsto \mathbf{P}\}$.) $[\,] \mapsto \mathbf{P}$ is the way of forcing each attribute C in list $\mathbf{P}$ to be a *constant*. Note that any table which satisfies $\mathcal{M}'$ satisfies $\mathcal{M}$. Once we find a swap, we halt in Algorithm 3.

Algorithm 3 TestOrderCompatible

Output: A result which states if $\mathbf{X}$ and $\mathbf{Y}$ are order compatible.

```
1: for i ← 0 to |X| − 1 do
2:   for j ← 0 to |Y| − 1 do
3:     result ← TestSingleOrderCompatible(i, j)
4:     if !result then
5:       return "false"
6: return "true"
```

Based on Definition 17, order compatibility for single attributes (over the attributes which are non-constant) is transitive for transitive domains. Therefore, we test if there is a *path* between A and Bin a graph consisting of the first not constant attributes from the left-hand side and a right-hand side of each UOD from $\mathcal{M}'$. We find this graph in Algorithm *FindOrderCompatibleGraph* (Algorithm 5). Finding a path by transitivity property over order-compatibility means that $\mathsf{A} \sim \mathsf{B}$ holds.

We assume the Algorithm *TestExistPath* which tests if there exists a path between two nodes. The problem of testing if there *exists* a path is simple. One can track the visited edges during the process of traversing the nodes. We can guarantee that each edge is visited only once. Hence, we can check the existence of the path in linear time. Note there is an edge per OD in $\mathcal{M}'$, so the number of edges (plus number of nodes) is $O(|\mathcal{M}|)$.

THEOREM 14. *(soundness and completeness) Algorithm 2 for testing logical implication $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$ for natural domains of UODs is sound and complete.*

Proof

Theorem 4 states that order dependency $\mathbf{X} \mapsto \mathbf{Y}$ holds *iff* $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

Case 1 $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$. We have already proven that testing $\mathbf{X} \mapsto \mathbf{XY}$ is sound and complete (Theorem 12).

Case 2 $\mathcal{M} \models \mathbf{X} \sim \mathbf{Y}$. This step is tested in algorithm *TestOrderCompatible* (Algorithm 3). If $\mathcal{M} \models \mathbf{X} \sim \mathbf{Y}$ is falsified, then we show that we are always able to construct a two-tuple table $\mathbf{t}$ which satisfies set of UODs $\mathcal{M}$ and has a *swap* (Definition 8) with respect to an UOD $\mathbf{X} \sim \mathbf{Y}$. The main body of this algorithm is a double-nested for-loop runs $|\mathbf{X}||\mathbf{Y}|$ times (and *terminates*). Inside, it invokes Algorithm 4 each time.

Algorithm 4 TestSingleOrderCompatible

Input: Attributes indexes i and j.

Output: "true" if single attributes LX[i] and LY[j] are order compatible, "false" otherwise.

```
1: if LX[i] = LY[j] then
2:   return "true"
3: else
4:   List P is a concatenation of lists LX.subList(0, i − 1)
     and LY.subList(0, j − 1)
5:   M' ← M ∪ {[ ] ↦ P}
6:   DEPEND ← TestFunctionalDependency([ ], M')
7:   if DEPEND[LX[i]] ∥ DEPEND[LY[j]] then
8:     return "true"
9:   else
10:    OC ← FindOrderCompatibleGraph
11:    return TestExistPath(LX[i], LY[j], OC)
```

Algorithm 5 FindOrderCompatibleGraph

```
1: initialize two dimensional array OC with [0; 0]
2: for l ← 0 to n − 1 do
3:   for k ← 0 to LS[l].size() − 1 do
4:     if DEPEND[LS[l][k]] = "false" then
5:       a ← DEPEND[LS[l][k]]
6:       for s ← 0 to RS[l].size() − 1 do
7:         if DEPEND[RS[l][s]] = "false" then
8:           b ← DEPEND[RS[l][s]]
9:           OC[l][0] ← a, OC[l][1] ← b
10:          break
11:      break
```

Algorithm 4 tests for each pair of attributes A and B from $\mathbf{X}$ and $\mathbf{Y}$, respectively, if it is possible to construct the described table $\mathbf{t}$ which has a swap between A and B, and which also satisfies $\mathcal{M}'$. $\mathcal{M}' = \mathcal{M} \cup \{[\,] \mapsto \mathbf{P}\}$, where $\mathbf{P}$ is a conjunction of lists prefixing A and B from lists $\mathbf{X}$ and $\mathbf{Y}$, respectively. Note that any table which satisfies $\mathcal{M}'$ satisfies $\mathcal{M}$. Therefore, by Definition 8 we enumerate through all the possible cases in the columns where an UOD $\mathbf{X} \sim \mathbf{Y}$ can by falsified by a *swap*. (It cannot be falsified by a *split*, Definition 7.)

We construct the table $\mathbf{t}$ (see Table 8) with values 0 and 1 only if both A and B are not *constants*. Attributes are partitioned into three groups: those that have the same values as A (and consequently swapped values to B), those that are constants; and the remaining attributes which have swapped attributes to A. Group A is the set of attributes which have a path with A in a data structure OC (in Algorithm 2.e., assigning the values in Algorithm 5). We use transitivity property for single attributes over order-compatibility. Algorithm *ExistPath* tests if there exists a path.

Table 8: Table $\mathbf{t}$.

Constants	A	B	Group A			Remaining attributes		
0	0	1	0	...	0	1	...	1
0	1	0	1	...	1	0	...	0

Table $\mathbf{t}$ satisfies $\mathcal{M}'$. Assume otherwise: for $\mathbf{M} \mapsto \mathbf{N} \in \mathcal{M}'$, $\mathbf{t}$ falsifies it. We do not introduce splits in table $\mathbf{t}$ that falsify $\mathbf{M} \mapsto \mathbf{MN}$ because by Theorem 12 the algorithm is sound and complete for inferring FDs. (Note that it applies also to constants which can be expressed as FDs.)

Consider $\mathbf{M} \sim \mathbf{N}$. Breaking this is the other way of falsifying UOD $\mathbf{M} \mapsto \mathbf{N}$ by Theorem 4. Let E be the first element which is not a constant from $\mathbf{M}$ and F from $\mathbf{N}$, respectively. If both E and F are from group A plus A or they are both from remaining attribute group plus B, then $\mathbf{M}$ and $\mathbf{N}$ order the tuples of $\mathbf{t}$ the same way. Therefore, E must be from one group and F from the other. Since the transitivity property holds over order-compatibility we would detect this. Contradiction. □

THEOREM 15. *(complexity for transitive domains) Testing logical implication for transitive domains of UODs, (that, is whether $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{Y}$) is solvable in polynomial time, $O(|\mathbf{X}||\mathbf{Y}||\mathcal{M}|)$*

Proof

By Theorem 4, $\mathbf{X} \mapsto \mathbf{Y}$ holds *iff* $\mathbf{X} \mapsto \mathbf{XY}$ and $\mathbf{XY} \leftrightarrow \mathbf{YX}$.

Case 1 Testing the logical implication that $\mathcal{M} \models \mathbf{X} \mapsto \mathbf{XY}$ can be done in $O(|\mathcal{M}|)$ time by Theorem 12.

Case 2 Algorithm 3 tests logical implication, that $\mathcal{M} \models \mathbf{X} \sim \mathbf{Y}$. In the main body of this algorithm, the double-nested for-loop runs $|\mathbf{X}||\mathbf{Y}|$ times invoking each time Algorithm 4. Algorithm 4 tests if each A from $\mathbf{X}$ is order compatible with each B from $\mathbf{Y}$. This is done by checking if there is a path in a graph. We keep track of visited edges. Hence, we can check if there is a path in linear time over $O(|\mathcal{M}|)$. Therefore the complexity of Algorithm 3 is $O(|\mathbf{X}||\mathbf{Y}||\mathcal{M}|)$. □

6. RELATED WORK

Ordered sets and lattices have been researched in mathematics [6]. Our concept of ODs is equivalent to *order-preserving mapping* between two *ordered* sets. The work in mathematics has concentrated on investigating the properties of, and relationships between, ordered sets rather than among the mappings. To the best of our knowledge, no complexity study for investigating relationship between mappings of ODs has been proposed.

Sorting is at the heart of many database operations: sort-merge join, index generation, duplicate elimination, ordering the output through the SQL order-by operator, etc. The importance of sorted sets for query optimization and processing has been recognized very early on. Right from the start, the query optimizer of System R [16] paid particular attention to *interesting orders* by keeping track of all such ordered sets throughout the process of query optimization. In [11] authors explored the use of sorted sets for executing nested queries. The importance of sorted sets has prompted the researchers to look *beyond* the sets that have been explicitly generated. Thus, [13] shows how to discover sorted sets created as generated columns via algebraic expressions. (In DB2, a generated column is a column that can be computed from other columns in the schema.) For example, if column A is sorted, so is the generated column G defined as $\mathsf{G} = \mathsf{A}/100 + \mathsf{A} - 3$ (that is, A $\rightsquigarrow$ G). [9] We show in [20] how to use relationships between sorted attributes discovered by reasoning over the physical schema.

Ordered dependencies were proposed in the context of database systems in [9]. However, the type of orders, hence the dependencies defined over them, were different from the ones we presented here. A dependency $\mathcal{X} \rightsquigarrow \mathcal{Y}$ holds if order over the values of *each* attribute of $\mathcal{X}$ implies an order over the values of *each* attribute of $\mathcal{Y}$. (In other words, the dependency is defined over the sets of attributes rather than lists.) Formally, an instance of a database satisfies a pointwise order dependency $\mathcal{X} \rightsquigarrow \mathcal{Y}$ if, for all tuples s and t, for every attribute A in $\mathcal{X}$, s_{A} *op* t_{A} implies that for every attribute B in $\mathcal{Y}$ s_{B} *op* t_{B}, where *op* $\epsilon\{<, >, \leq, \geq, =\}$. In [9] a sound and complete set of inference rules for such dependencies is defined with demonstrating that determining logical implication is co-NP-complete. A practical application of the dependencies for an improved index design is presented in [7]. A novel integrity constraint for ordered data, sequential dependencies which defined also over *sets* of attributes was introduced in [10]. For example, a sequential dependency $\mathsf{sequence\_id} \rightsquigarrow_{[5,6]} \mathsf{time}$ means that time *gaps* between consecutive sequence numbers are between 5 and 6. The authors present a framework for discovering which subsets of the data obey prescribed sequential dependencies.

Dependencies defined over lexicographically ordered domains were introduced in [14] under the name *lexicographically ordered functional dependencies*. The paper [15] by the same author develops a theory behind both lexicographical as well as pointwise dependencies. (The latter were simpler than the dependencies defined in [9].) A set of inference rules (proved to be sound and complete) is introduced for pointwise dependencies, but not for lexicographical dependencies in this work. Only a chase procedure is defined for the latter, for which the order dependencies are defined as we do in this paper. (We call these UODs.) Interestingly, the complexity of testing logical implication for ODs has also not been studied, which is the subject of our work. Recently, in [19] we presented a sound and complete axiomatization for UODs. UODs do not consider bidirectionality (a mix of `asc` and `desc`) as ODs which we introduced for the first time in [18]. Many times relations prove to be *nearly-sorted*; most of tuples are close to their place in the order. An interesting study of establishing whether a given stream is sufficiently nearly-sorted was described in [4].

7. CONCLUSIONS

Ordering permeates databases, to such an extent that we take it for granted. We expect it to be exploited wisely in query plans. It is requested by many queries but is relatively expensive to perform. The goal of this paper was to develop a theory behind the complexity of ODs. To the best of our knowledge, this is the first attempt to study the complexity of such dependencies.

We devise a chase procedure for testing logical implication for ODs and show that the inference problem for ODs and INDs is undecidable. We present that testing logical implication for UODs is co-NP-complete, as well as FDs over ODs. Therefore, they are not amenable to fast algorithmic solutions. However, we demonstrated that testing logical implication of FDs over UODs is linear. Finally, we have also shown a transitive domain over which the inference problem for UODs is tractable. We designed a polynomial algorithm for testing logical implication.

There is more that can be done, and that we plan to do. Future work in this area should pursue two lines of research: further investigation of the theoretical questions; and, appli-

[9] We use the arrow "$\rightsquigarrow$" for simplicity for different type of orders, regardless.

cations of the theoretical framework in a practical database setting. These are further things we plan to do.

- We plan to work on extending our work for axiomatization for UODs [19] into axiomatization for ODs, which allow the mix of ascending and descending orders. Such axiomatization might provide insight into how ODs behave, and provide input for useful query rewrites.
- One of the practical applications which we are currently working on is a *sound* theorem prover. We prove in this work that testing logical implication for ODs is co-NP-complete. However, it is the lack of transitivity over the order-compatibility that is at the heart of of the high complexity. That is why the *Chain* axiom is necessary for the sound and complete axiomatization of UODs (Figure 1). We would like to investigate if there is a polynomial algorithm for reasoning over the first five axioms, excluding Chain axiom (Figure 1). Such a theorem prover would be a useful tool in query optimization and an alternative approach to what we proposed in this work. (We define a domain property that makes reasoning over ODs efficient.)
- We are working on introducing a framework for discovering *conditional order dependencies*. (Conditional sequential dependencies were proposed in [10].) A conditional order dependency can be represented as a pair ($\mathbf{X} \mapsto \mathbf{Y}$, $\mathbf{T}_r$), where $\mathbf{X} \mapsto \mathbf{Y}$, refereed to as the embedded OD, and $\mathbf{T}_r$ is a range pattern tableau defining over which rows the dependency applies. It would be a novel integrity constraint allowing one to express that an OD $\mathsf{date} \mapsto \mathsf{salary}$ holds within a given $\mathsf{employee\_id}$.
- If $\mathsf{ABC} \mapsto \mathsf{D}$ holds but not $\mathsf{AB} \mapsto \mathsf{D}$, is ordering by AB useful if we need a stream sorted by D? If the stream is sorted by AB, it may be *nearly sorted* on D. If it were known that every partition of AB is small, each AB-partition could be sorted on-the-fly in main memory, removing the need for an external sort operator. We believe the work of [4] and this work on order dependencies could be combined to formalize the concept of nearly sorted.
- We are exploring the use of ODs for *database design* [3]. The concept of functional dependency lies at the heart of database design and the relational model. Order dependency extends functional dependency in a quite natural way to include also semantics of order over the data. ODs can reveal redundancies that cannot be detected using FDs alone. This leads one to wonder about the concept of normalization modulo ODs. It would be an interesting research topic to extend the results obtained there to the design of relational databases.

Acknowledgments.
We thank Wenbin Ma from IBM laboratory in Toronto for his encouragement and helpful suggestions throughout the project.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. In *Addison-Wesley, 157-211*, 1995.

[2] W. Armstrong. Dependency Structures of Database relationships. In *Proceedings of the IFIP Congress, 580-583*, 1974.

[3] C. Beeri and P. Bernstein. Computional Problems Related to the Design of Normal Form Relational Schemas. *TODS 4(1):30-59*, 1979.

[4] S. Ben-Moshe, Y. Kanza, E. Fischer, A. Matsliah, M. Fischer, and C. Staelin. Detecting and exploiting near-sortedness for efficient relational query evaluation. In *ICDT, 256-267*, 2011.

[5] P. Bernstein. Synthesing third normal form relations from functional dependencies. *TODS, 1(4): 277-298*, 1976.

[6] B. Davey and H. Priestley. Introduction to Lattices and Order. In *Cambridge University Press, 1-50*, 2002.

[7] J. Dong and R. Hull. Applying Approximate order dependency to Reduce Indexing Space. In *SIGMOD, 119-127*, 1982.

[8] M. Garey and D. Johnson. A Guide to NP-completness. In *Freeman*, 1979.

[9] S. Ginsburg and R. Hull. Order dependency in the Relational Model. *TCS, 26(1): 149-195*, 1983.

[10] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential Dependencies. *PVLDB, 2(1): 574-585*, 2009.

[11] R. Guravannavar, H. Ramanujam, and S. Sudarshan. Optimizing Nested Queries with Parameter Sort Orders. In *VLDB, 481-492*, 2005.

[12] R. Kimball and M. Ross. The Data Warehouse Toolkit Second Edition. The Complete Guide to Dimensional modeling. In *John Wiley and Sun, 217-227*, 2012.

[13] T. Malkemus, P. S., B. Bhattacharjee, and L. Cranston. Predicate Derivation and Monotonicity Detection in DB2 UDB. In *ICDE, 939-947*, 2005.

[14] W. Ng. Lexicographically Ordered Functional Dependencies and Their Application to Temporal Relations. In *IDEAS, 279-287*, 1999.

[15] W. Ng. An Extension of the Relational data model to incorporate ordered domains. *TODS, 26(3) 344-383*, 2001.

[16] P. Selinger and M. Astrahan. Access Path Selection in a Relational Database Management System. In *SIGMOD, 23-34*, 1979.

[17] D. Simmen, E. Shekita, and T. Malkemus. Fundamental Techniques for Order Optimization. In *SIGMOD, 57-67*, 1996.

[18] J. Szlichta, P. Godfrey, and J. Gryz. Chasing Polarized Order Dependencies. In *AMW,168-179*, 2012.

[19] J. Szlichta, P. Godfrey, and J. Gryz. Fundamentals of Order Dependencies. *PVLDB, 5(11): 1220-1231*, 2012.

[20] J. Szlichta, P. Godfrey, J. Gryz, W. Ma, P. Pawluk, and C. Zuzarte. Queries on Dates: Fast Yet not Blind. In *EDBT, 497-502*, 2011.

[21] J. Szlichta, P. Godfrey, J. Gryz, W. Ma, W. Qiu, and C. Zuzarte. Business-Intelligence Queries in DB2 with Order Dependencies. Technical report, York University, 2012. Submitted for review in ICDE. www.cse.yorku.ca/techreports/2012/CSE-2012-04.pdf.